

Intermediate Velocity Training

Part 1

String Manipulation

JAVA METHODS TO USE ON STRINGS



How to Dynamically Inspect and Transform Strings

CONTROL YOUR STRINGS

```
61 #set( $multiSelectField = $velocityReportUtil.getValueForField($doc, "multi_select_field_name", $dateFormat))
62
63 ## Add a space after each comma separated value
64 #set( $multiSelectField = $multiSelectField.replaceAll(",", " , "))
65
66 <p class=MsoNormal>${multiSelectField}</p>
```

- Most data supplied to your Velocity report from your Jama Connect instance will be Strings, this also includes rich text fields which are provided as a String block of HTML code.
- You can leverage Java methods to manipulate and validate these String values, such as:

`.replaceAll()`, `.split()`, `.contains()`, `.indexOf()`, `.length()`

- Or convert a non-string value to a String:

`.toString()`

Strings Continued...

- As an example, let's say our Jama Connect items hold a placeholder called "[OPERATING SYSTEM]" in which we want to replace it with a user input report String parameter.

NAME:

[OPERATING SYSTEM]

DESCRIPTION:

Many hearing aid on the market require a pendant or other device to transfer sound between the hearing aid and the [OPERATING SYSTEM]. If we want to increase market share among younger patients in Asia, primarily Japan, we need to find a way to directly integrate with [OPERATING SYSTEM].

Parameters

Enter OS:

SymbianOS

- To accomplish this report requirement, we can utilize the following Java String methods of .length(), .toString(), and .replaceAll() ...

Strings Continued...

```
87  #set( $replaceText = false)
88
89  ## Check if the parameter exists and what length the String is
90  #if( $reportOS && $reportOS.length() > 0 )
91      #set( $replaceText = $reportOS )
92  #end
93
94  ## Ensure document's name is of String type
95  #set( $docName = $doc.name.toString())
96
97  ## Clean the rich text String description field
98  #set( $desc = "#formatRichText( $doc.description )")
99
100  #if( $replaceText )
101
102      ## Escape the square brackets in the pattern as they are special characters in regex
103      ## Replace the placeholder [OPERATING SYSTEM] with the value of $replaceText
104      #set( $docName = $docName.replaceAll("\[OPERATING SYSTEM\]", "$replaceText"))
105      #set( $desc = $desc.replaceAll("\[OPERATING SYSTEM\]", "$replaceText"))
106
107  #end
```

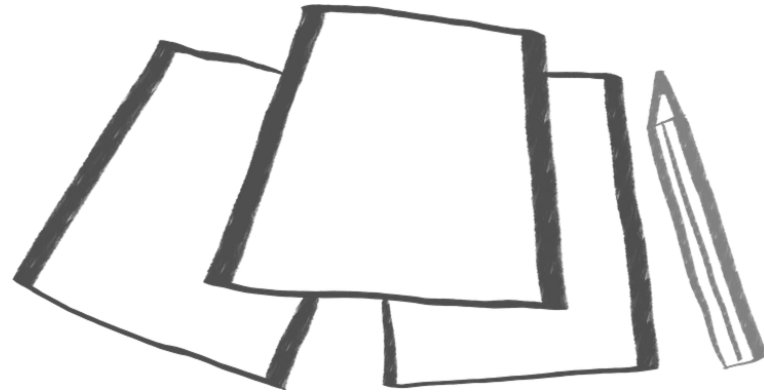
➤ Example Output where “[OPERATING SYSTEM]” has been replaced with the parameter value of ‘SymbianOS’:

1 SymbianOS

Many hearing aid on the market require a pendant or other device to transfer sound between the hearing aid and the SymbianOS. If we want to increase market share among younger patients in Asia, primarily Japan, we need to find a way to directly integrate with SymbianOS.

Page Layout Switch

PAGE LAYOUT CHANGE AND @PAGE RULES



@page Rule(s) and Declarations

UNIQUE PAGE LAYOUTS AND THEIR DEFINITIONS

- In your Velocity reports and converted mockup templates you will find @page rule(s) which define your page container such as page size, margins, applied header / footer etc...
- Each @page rule will have a unique name and associated div style class that applies that definition using the 'page' CSS property.
- Using these div elements with your applied page definition your report will follow the defined @page rules to all content encapsulated.
- In which you can have multiple @page definitions and switch between them to meet your reporting needs. However, you typically only need a Portrait and Landscape option.

```
<style>
<!--
@page WordSectionPortrait
{size:8.5in 11.0in;
margin:1.0in 1.0in 1.0in 1.0in;
mso-header-margin:.5in;
mso-footer-margin:.5in;
mso-title-page:yes;
mso-header:pageHeader;
mso-footer:pageFooter;
mso-first-header:titlePageHeader;
mso-first-footer:titlePageFooter;}
div.WordSectionPortrait
{page:WordSectionPortrait;}
@page WordSectionLandscape
{size:11.0in 8.5in;
mso-page-orientation:landscape;
margin:1.0in 1.0in 1.0in 1.0in;
mso-header-margin:.5in;
mso-footer-margin:.5in;
mso-title-page:yes;
mso-header:pageHeader;
mso-footer:pageFooter;
mso-first-header:titlePageHeader;
mso-first-footer:titlePageFooter;}
div.WordSectionLandscape
{page:WordSectionLandscape;}
-->
</style>
```

Switching Page Layout

```
<div class=WordSectionPortrait>
    #responsiveHeaderFooter() ## Generate headers and footers
    ## {content ...}
</div> ## Close the Portrait page definition

<br clear=all style='page-break-before:always;mso-break-type:section-break'>

<div class=WordSectionLandscape>
    #responsiveHeaderFooter() ## Generate headers and footers
    ## {content ...}
</div> ## Close the Landscape page definition
```

Steps

- 1) Determine where your page definition should change in the report <body>...</body>.
- 2) Close the current div page definition using a closing </div>.
- 3) Following your closing </div> add a
 break tag with 'mso-break-type: section-break' applied as shown above.
- 4) Open a new div with your alternative page definition <div...>
- 5) Call the #responsiveHeaderFooter() macro following your new page definition <div...> to ensure they are applied to the new page layout.

- ❖ **Tip 1:** We recommend utilizing the percentage unit for your width definitions on your headers and footer to ensure responsive resizing to the page dimensions.
- ❖ **Tip 2:** Be aware that opening a new page definition applies your header and footer in which you should always double check that your page properties are set to ensure that any unexpected behavior is prevented such as mso-title-page being set to 'no'.

Data Collections and Trace Levels

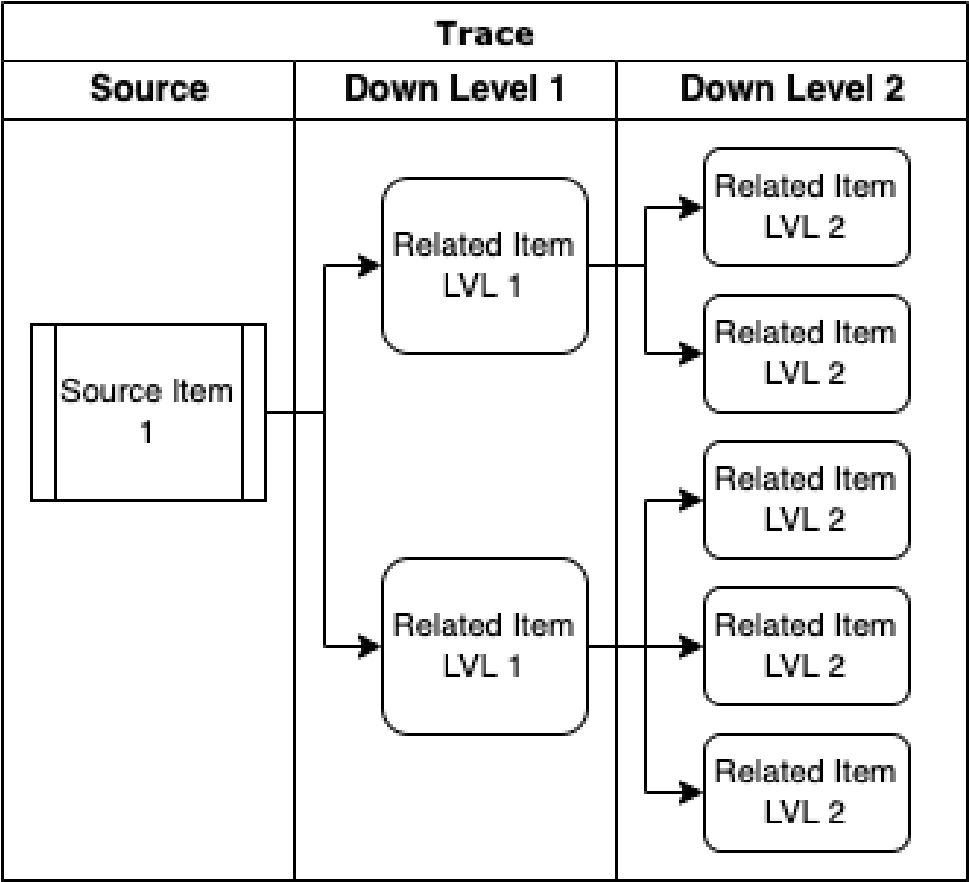
OBJECT LISTS, MULTI-TRACE RELATIONSHIPS, AND TABLE MERGE CELLS



Gathering Multi-Level Related Items

BUILD LISTS OF UPSTREAM/DOWNSTREAM RELATIONSHIPS BY LEVEL

- In the Basic Velocity Training we covered how to extract related items 1 level upstream or 1 level downstream from the source item. To gather more than 1 level we must gather 1 level at a time or use retrieval nesting.
- The quantity of related items can increase exponentially for each level you traverse from the source item.
- As you can see in the diagram representing items gathered from 2 levels downstream from the 1 source item the report cycles, extracts, and gathers data from a total of 7 items.
- To represent this structure in a trace report output we will utilize two core concepts: object lists & merged cell format structure.



Object List Construction

ORGANIZE YOUR DATA INTO A COLLECTION OF OBJECTS

```
$downstreamTrace = [ { "Src": $Src, "L1": $L1, "L2": $L2s }, { "Src": $Src, "L1": $L1, "L2": $L2s } ... ]
```

The diagram illustrates the construction of an object list. It shows a variable `$downstreamTrace` assigned to an array. The array contains two objects, indexed 0 and 1. Each object has three key-value pairs: `"Src": $Src`, `"L1": $L1`, and `"L2": $L2s`. The objects are enclosed in curly braces, and the array is enclosed in square brackets. Ellipses indicate that the array can contain more objects.

- The ArrayList structure can hold a dynamic list of object definitions which is an extremely valuable tool in custom reporting.
- You can use object lists to join, organize, and associate data from multiple sources into a unified, ordered format that fits your reporting or display logic.
- Using the `.add()` method on an ArrayList type variable, you can store custom-defined objects which contain key-value pairs that support flexible data extraction.
- A common utilization of this structure is for handling multi-level trace data.

Construct Trace Data Object List

```
108  ## Initiate ArrayList to add relationship objects to
109  #set( $downstreamTraceList = [] )
110
111  #getRelationships( $doc true ) ## true - downstream
112  #set( $L1s = $returnRels )
113
114  #foreach( $dnRel in $L1s )
115
116      #set( $levelTwos = [] ) ## Initiate ArrayList to add Level 2 items to
117      #set( $L1 = $dnRel.toDocument )
118
119      #getRelationships( $L1 true ) ## Foreach Level 1 item get the Level 2 Items
120      #set( $L2s = $returnRels )
121
122      #foreach( $dn2Rel in $L2s )
123          #set( $L2 = $dn2Rel.toDocument )
124          #set( $success = $levelTwos.add($L2) ) ## Add the Level 2 item to the $levelTwos
125      #end
126
127  ## Add the relationship object to the $downstreamTraceList
128  #set( $success = $downstreamTraceList.add({"Src":$doc, "L1":$L1,"L2s":$levelTwos}))
129
130  #end
```

Access Object List Trace Data

OBJECT INDEX AND OBJECT KEY

- Since the object list structure is an instance of ArrayList, we can access specific objects by their index using either array-style syntax [0] or the .get(index) method.
- If you want to extract the values that are held by the object you will need to first extract the object, then access each value using the Key identifier via dot notation or the .get() method.

```
135  ## Access an object by index position
136  #set( $firstObjectInList = $downstreamTraceList.get(0))
137
138  ## You can do the same thing with slightly different syntax
139  #set( $firstObjectInList = $downstreamTraceList[0])
140
141  ## Access the key values with dot notation
142  #set( $L1 = $firstObjectInList.L1 )
143  #set( $L2s = $firstObjectInList.L2s )
144  #set( $Src = $firstObjectInList.Src )
145
146  ## Access each object in the list by iterating over them
147  ▼ #foreach( $traceObj in $downstreamTraceList )
148      ## You can also use this syntax to get key values
149      #set( $L1 = $traceObj.get("L1"))
150      #set( $L2s = $traceObj.get("L2s"))
151      #set( $Src = $traceObj.get("Src"))
152      ...
153  #end
```

Construct Merged Cell Trace Table

DYNAMICALLY BUILD ROWS WITH CUSTOM ROWSPAN

- Utilizing our object list structure which organizes our trace items by trace level we can efficiently build our merged cell trace table.
- Moving left to right in the trace table the rowspan value of the trace level item will to be equal to the max count of items per level downstream from it, for example using the 'Source' level:

```
$sourceRowSpan = $mathTool.max( $numL1s, $numL2s, 1)
```

- In this example for a given 'Level 1' item the rowspan will simply be the count of how many 'Level 2' items downstream from it.
- As you build the dynamic rows in your trace table you can strategically calculate these rowspans and apply them to the table cell.

Source	Level 1	Level 2

Source	Level 1	Level 2
rowspan=5	rowspan=2	
	rowspan=3	

Source	Level 1	Level 2
T-UN-1	T-SR-1	T-VER-1
		T-VER-2
	T-SR-2	T-VER-3
		T-VER-4
		T-VER-5

Build Trace Rows Dynamically

CREATE MERGE CELLS VIA ROWSPAN

```
159 #foreach($traceObj in $downstreamTraceList)
160   #set($L1Item = $traceObj.get("L1"))
161   #set($L2Items = $traceObj.get("L2s"))
162
163   <table>
164     #set($L1RowSpan = $L2Items.size())
165     #if($L1RowSpan == 0)
166       #set($L1RowSpan = 1)
167     #end
168
169     #if( $L2Items.size() > 0 )
170       #foreach($L2Item in $L2Items)
171         <tr>
172           #if($foreach.first)
173             <td rowspan=$L1RowSpan>$L1Item.documentKey - $L1Item.name</td>
174           #end
175
176           <td>$L2Item.documentKey - $L2Item.name</td>
177         </tr>
178       #end
179     #else
180       <tr>
181         <td rowspan=$L1RowSpan>$L1Item.documentKey - $L1Item.name</td>
182         <td>N/A</td>
183       </tr>
184     #end
185   </table>
186 #end
```

- This example shows how to build a two-column merged trace table of Level 1 and Level 2 downstream trace items.
- The L1Item cell rowspan is set to the size of how many L2Items are downstream from it.
- If an L1Item does not have any downstream items, then its rowspan is set to 1 because we still want to build a row for that L1Item.
- It's important to grasp how to properly calculate and apply your rowspan properties when building merged cell format before advancing to complex trace structures.

Bonus Data Structure: Map

ANOTHER DATA STRUCTURE FOR BUILDING COLLECTIONS

```
53  ## Initiate a LinkedHashMap
54  #set( $downstreamTraceMap = {})
55
56
57  ## Utilize the Java method .put to add a key and value to the Map
58  #set( $success = $downstreamTraceMap.put("$sourceItem.id", [$sourceItem, [[L1Item,L2Items],...]]))
59
```

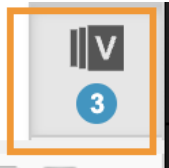
- A LinkedHashMap is a Java object structure that stores key-value pairs in a single collection, while maintaining the insertion order of the entries.
- Using this structure, you can create an efficient key-based lookup system for your data, which is useful for various custom reporting implementations.
- To build a map you use the .put() method, which takes both a key and a value.
- If the key already exists in the map, its associated value will be updated with your latest entry. If the key does not exist, the new key-value pair will be added to the map.

Document Versions

ACCESS ITEM VERSIONS AND ASSOCIATED BASELINES / REVIEWS



Viewing Item Versions



From	To	Version	Baselines / Reviews	Change details	User comment and transitions	Created	By
☐	☐	3	Review Text: Test v1 Baseline Review	"Rationale" changed from "" to "fdsfsdfsdfs"		04/14/2025 04:32:23 pm	Sample User
☐	☐	2		"Description" changed		04/14/2025 04:32:07 pm	Sample User
☐	☐	1		Initial Creation		04/14/2025 04:31:53 pm	Sample User

- In Jama Connect when viewing an individual document (Single Item View) you can utilize the Version widget button to view all versions of the document in a table format as shown above.
- The provided table columns outline various information about each version such as 'Version', 'Baselines / Reviews', 'Change details', 'User comment', 'Created', and 'By' that is accessible to us through the Version Interface on the Velocity API.

Is the Document active?

THE MOST CURRENT VERSION IS THE ACTIVE DOCUMENT

- When it comes to document versions the most current item version is referred to as the 'Active' document while all prior versions are 'Inactive'.
- Be aware that baselines can hold prior document versions that differ from the most current version displayed on your project tree.
- To validate this, you can utilize \$doc.active to confirm 'true' that the document is the most current or 'false' that the item is a prior item version or that it has been deleted.

The screenshot displays the Jama Software interface for a document titled "Surgical Installation". A dropdown menu is open, showing a list of versions from "Version 1" to "Version 7 (most current)". The "Version 7 (most current)" option is highlighted with an orange underline. To the right of the dropdown, the text "\$doc.active == false" is displayed in orange. Below the dropdown, the "PROJECT ID:" is shown as "MDSD-SR-2".

Surgical Installation V7 ▼

System Requirement • Mod Version 7 (most current)

\$doc.active == true

Version 6

Version 5

Version 4

Version 3

Version 2

Version 1

\$doc.active == false

PROJECT ID:

MDSD-SR-2

Get All Versions of a Document

SORT THE VERSIONS WITH THE \$sortTool

```
#set( $thisDocVersions = $documentSource.getAllVersionsByDocumentId($mathTool.toInteger($doc.id)) )
#set( $thisDocVersions = $sortTool.sort($thisDocVersions, "createdDate:asc") )
```



Note: The \$sortTool is designed to be used within a #foreach loop but can be used like this too.



For ascending
:asc
For descending
:desc

- **Retrieving All Document Versions:** You can extract all document versions by utilizing the `.getAllVersionsByDocumentId()` method available on the Velocity API Interface DocumentSource. This method returns an ArrayList of Version Interface type objects containing each document version.
- **Sorting Document Versions:** At times you may need to sort the document versions in which the \$sortTool can be used. In the above example we are sorting the document's versions by their created date in ascending order.

Interface Version Method Descriptions

Method	Description
<code>.getBaseLines()</code>	Retrieve all baselines and revisions that include the specific Version of the document, corresponds with the 'Baselines / Reviews' column of the Version dashboard UI.
<code>.getComments()</code>	Retrieve the Version comment changes, corresponds with the automatically generated change details in the 'Change details' column of the Version dashboard UI.
<code>.getDocument()</code>	Access document version field information.
<code>.getOriginDocument()</code>	Traverse to the most current Version, 'Active' document.
<code>.getUserComment()</code>	Access user comments applied by the person who made the changes to the Version, corresponds with the 'User Comment and transitions' column of the Version dashboard UI.
<code>.getVersionNumber()</code>	The Version Number, corresponds with the 'Version' column of the Version dashboard UI.

Item Version Data

VERSION DATA & DOCUMENT VERSION FIELD INFORMATION

```
148  ## The $sortTool is utilized within a #foreach loop
149  #foreach( $version in $sortTool.sort($thisDocVersions,"createdDate:asc") )
150
151      ## set the document object, which returns a ContourItem particular to
152      ## the Version iterated on
153      #set( $docByVersion = $version.document )
154
155      ## Print the document name, the version number, and a custom field per document version
156      <p>$docByVersion.name v$!version.versionNumber</p>
157      #set( $rationale = "" )
158      #set( $rationale = $velocityReportUtil.getValueForField($docByVersion, 'rationale', $dateFormat))
159      #formatRichText($!rationale)
160
161  #end
```

- When cycling through the document versions ArrayList you have access to data that pertains to each version itself.
- To access the field values of the document at a particular version you need to apply the .getDocument() method. This will return an Interface ContourItem type object particular to the version you are iterating on which will hold the field values that existed on said document version.

Access Baselines Per Version

A VERSION MAY BE INCLUDED IN NUMEROUS BASELINES

- As you will see in your document Version widget dashboard in the Jama Connect UI when a document is baselined, it will be associated to that document version.
- **Extract Version Baselines:** From the Interface Version you can utilize the .getBaseLines() method to retrieve an ArrayList of all baselines associated with your document version.
 - Note the capitalized 'L' in the baseLines method call.

Interface Version

getBaseLines()

```
#foreach( $version in $thisDocVersions )
  #set( $docVersion = $version.document )
  #set( $baselines = $version.baseLines)

  #if( $baselines.size() > 0 )
    #foreach( $baseline in $baselines )
      ...
    #end
  #end
#end
```

Access Review Revisions Per Version

TRAVERSE FROM THE INTERFACE BASELINE TO THE INTERFACE REVISION

- **Extract Version Reviews:** From the Interface Version you can utilize the following method path to retrieve any reviews associated with your document version.
 - Extract your versions associated baselines using `.getBaseLines()`
 - From each extracted baseline call `.getRevisionWithBaseline()` which provides a review Revision
 - From your revision you can extract the corresponding Review which is associated to your document version

Interface Version
`getBaseLines()`



Interface Baseline
`getRevisionWithBaseline()`



Interface Revision
`getReview()`



Interface Review
`getName()`

Next Steps?



HOMework

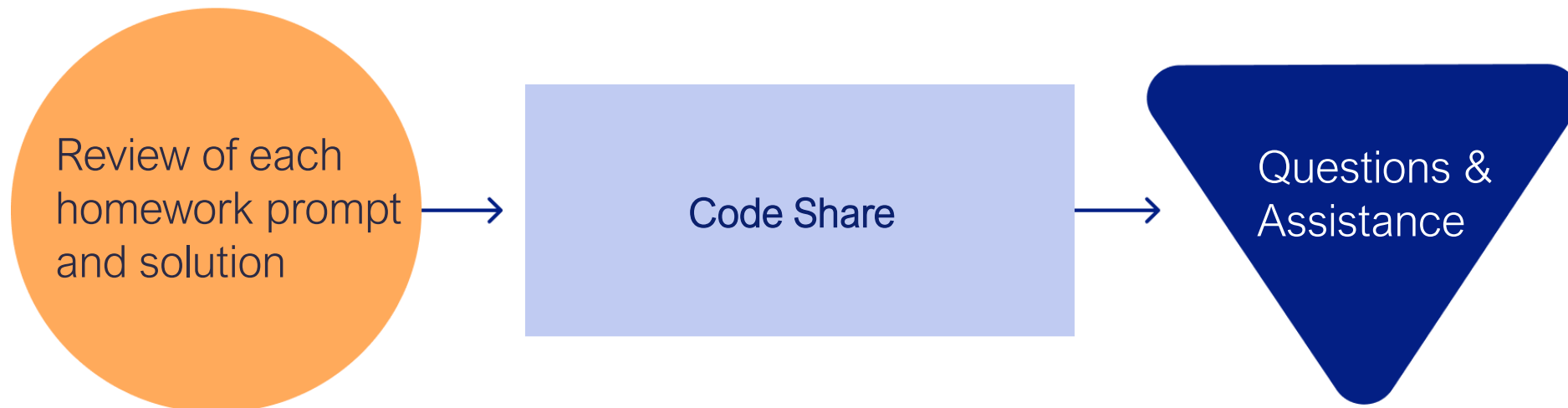
COMPLETE THESE PROMPTS BEFORE PART 2

- In the Intermediate Customer Part 1 Training Materials folder please find these homework files:
 - *Intermediate_Training_Homework_Prompts.docx*
 - *Intermediate_Training_Mockup_Example.docx*
 - *Intermediate_Training_Template.vm*
- Follow the provided instructions found in the *Intermediate_Training_Homework_Prompts.docx* file which outlines defined exercises that utilize the topics covered in this training.
- Convert the provided *Intermediate_Training_Mockup_Example.docx* to a Single File Webpage .mht file to be used in your homework exercises as directed.
- If you get stuck on any prompt, make a note of it to discuss with your assigned developer in the Part 2 session and move onto the next exercise.

Live Session Training Part 2

WHAT TO EXPECT

- Part 2 will entail a 1-hour live session that will begin of a review of the provided homework prompts.
- It is expected that you complete these tasks to the best of your ability to both practice and get the most benefit from the live session.
- Following the homework review, participants will have opportunity to share their code.
- Any remaining time will be dedicated to further questions and assistance on any present blockers.



EESR Customer Modified Report Policy

In the event that a customer decides to modify a Jama-developed custom report, any resulting defects are not covered under the 3-month defect warranty period provided upon engagement delivery. Customers can contact their Jama Software Consultant or CSM to submit a billable request ticket in order to resolve any caused defects, if required. Customers will be expected to provide their modified report code file to our team or to request the original Jama-developed report be updated with their reporting needs. Any customer-modified reports will be subjected to higher scoping estimates to account for the additional time required to review or reimplement the added code in order to ensure they meet custom reporting standards. All requests for customer-modified Jama-developed reports remain subjected to our team's normal backlog timeline.